Before you start:

Homework Files

You can download the starter files for coding as well as this *tex* file (you only need to modify *homework2.tex*) on canvas and do your homework with latex. Or you can scan your handwriting, convert to pdf file, and upload it to canvas before the due date. If you choose to write down your answers by hand, you can directly download the pdf file on canvas which provides more blank space for solution box.

Submission Form

A pdf file as your solution named as VE281_HW2_[Your Student ID]_[Your name].pdf uploaded to canvas
Estimated time used for this homework: **3-4 hours.**

0 Student Info

Your name and student id:

Solution: Wang Zi Ning 520370910042

1 Selection Algorithms (14 points)

(a) For random selection algorithm, in what cases do we encounter the worst-case or the best-case scenario? Is it related to the input sequence? What are the respective run-time under these circumstances? (7 points)

Solution: The worst case is that each time the distance (if sorted) between the target and the pivot is the maximum. For example, the target is 1th, but the pivot is the last. The target is the last but the pivot is the 1th. The target is the median one but the pivot is the 1st or last.

The best case is that the first pivot is exactly the target. For example, the target is the second and the pivot is exactly the second one.

They are not related to the input sequence as each element have the same possibility to become the pivot in theory.

The worst case needs to choose pivot $\Theta(n)$ times and each pivot need $\Theta(n)$ to check its order.

The best case needs to choose pivot $\Theta(1)$ times and each pivot need $\Theta(n)$ to check its order.

The worst case is $\Theta(n^2)$, where n is the length of the sequence. The best case is $\Theta(n)$.

(b) As for deterministic selection algorithm, we have discussed about the divide-into-groups-of-5 strategy in class. What about we divide the sequence into groups of 7? Find out the recurrence relation as well as the worst-case time complexity of this approach. What are the advantages and disadvantages of having greater group size (i.e. from 5 to 7)? (7 points)

Solution: We have $T(n) = T(\frac{n}{7}) + T((1 - \frac{1}{2} \times \frac{4}{7})n) + O(n) = T(\frac{n}{7}) + T(\frac{5n}{7}) + O(n)$.

Also we can get $T(n) \leq 7cn = O(n)$. The worse case also ensure $O(n)$.

Having greater group size can reduce the time to get median of medians (from 10 to 7) but increase the time to get medium in each group(c will increase). Having greater group size will only change the constant number and will not have great changes when n is quite large.

2 Hashing Zoo (29 points)

Suppose Prof. Blue Tiger is using a hash table to store information about the grades of his students. The keys are strings and the values are integers. Furthermore, he uses a very simple function t where the hash code of a string is the integer representing its first letter. For example:

- t("Blue Tiger") = 1
- t("Red Flamingo") = 17
- t("Glass Frog") = 6

And we have:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Now, assume we are working with a hash table of size 10 and the hash function $h(t) = t \ \% \ 10$. This means that "Red Flamingo" would hash to 17, but ultimately fall into bucket $17 \ \% \ 10 = 7$ of our table. For this problem, you will determine where each of the given name lands after inserting a sequence of values using three different collision resolution schemes:

- linear probing
- quadratic probing
- double hashing with $h_i(t) = h(h(t) + ((16 - t) \ \% \ 6) * i)$

For each of these three collision resolution schemes, determine the resulting hash table after inserting the following (*key*, *value*) pairs in the given order:

1. ("Blue Tiger", "100")
2. ("Gold Monkey", "65")
3. ("Red Flamingo", "88")
4. ("Glass Frog", "96")
5. ("Rainbow Horse", "80")
6. ("Honeydew Alligator", "70")
7. ("Pink Elephant", "101")

Every incorrect value counts for 1 point.

2.1 Linear Probing (7 points)

Please use the **linear probing** collision resolution method to simulate the given insertion steps, and then show the final position of each (*key*, *value*) pair inside the related buckets below.

Solution: Write cpp code to simulate insertion and get the result.

Index	0	1	2	3	4
Key	"Honeydew Alligator"	"Blue Tiger"			
Value	70	100			
Index	5	6	7	8	9
Key	"Pink Elephant"	"Gold Monkey"	"Red Flamingo"	"Glass Frog"	"Rainbow Horse"
Value	101	65	88	96	80

2.2 Quadratic Probing (7 points)

Please use the **quadratic probing** collision resolution method to simulate the given insertion steps, and then show the final position of each (*key*, *value*) pair inside the related buckets below.

Solution: Write cpp code to simulate insertion and get the result.

Index	0	1	2	3	4
Key	"Glass Frog"	"Blue Tiger"		"Honeydew Alligator"	
Value	96	100		70	
Index	5	6	7	8	9
Key	"Pink Elephant"	"Gold Monkey"	"Red Flamingo"	"Rainbow Horse"	
Value	101	65	88	80	

2.3 Double Hashing (7 points)

Please use the **double hashing** collision resolution method to simulate the given insertion steps, with the double hash function $h_i(t) = h(h(t) + ((16 - t) \ \% \ 6) * i)$, and then show the final position of each (*key*, *value*) pair inside the related buckets below.

Solution: Write cpp code to simulate insertion and get the result.

Index	0	1	2	3	4
Key	"Glass Frog"	"Blue Tiger"		"Honeydew Alligator"	
Value	96	100		70	
Index	5	6	7	8	9
Key	"Rainbow Horse"	"Gold Monkey"	"Red Flamingo"	"Pink Elephant"	
Value	80	65	88	101	

2.4 Possible Insertion Order (8 points)

Suppose you have a hash table of size 10 storing Blue Tiger's family members' favorite letter which shares the same t(key) and h(t). It uses open addressing with linear probing. After entering six values into the empty hash table, the state of the table is shown below.

Index	0	1	2	3	4	5	6	7	8	9
Key					O	P	G	Q	H	F

How many insertion orders are possible? Explain your answer clearly.

Solution: t(O)=14, t(P)=15, t(G)=6, t(Q)=16, t(H)=7, t(F)=5. So h_0 is applied for O,P,G. h_1 is applied for Q,H. h_4 is applied for F.
So Q is after G. H is after Q. F is after H,Q,G,P.
Then we can get the order that GQHF.
O can be any place and P is in front of F.
So there is 4 possible for G,Q,H,F,P and $4 \times 6 = 24$ possible totally.

3 Basic Hashing Analysis (13 points)

3.1 Lazy Deletion (4 points)

In the lectures, we have mentioned one way to implement **remove()** which is simply marking the bucket as *deleted*. This method is easy to understand and implement. However, do you think there is any backwards of this implementation of **remove()**?

Solution: Firstly, more space is needed to store the flag. Also, time will increase to find an key if there is a lof of **remove()** called before.

3.2 Alternative for Remove (9 points)

Besides the usage of *deleted*, Blue Tiger thinks of another approach to improve the performance. The strategy he proposes is that when we need to remove a certain element x in hash table, we first find the last element y in the cluster where x locates. Then we swap x and y, and simply delete x.

Do you think this strategy will work? If not, please propose a way to fix this strategy without marking the bucket as *deleted*. Suppose linear probing is used to avoid collision.

Solution: This strategy will bot work.
If the table is not full, which means there exist an end for the cluster of x, to remove x ($h_i(x) = a$), we need to find (from the last element in the cluster) y at b which fit both $h_j(y) = b$ and $h_k(y) = a, k < j$ and swap x and y. Then we need to remove x at the new place following the above strategy recursively until there is no y in the cluster. Finally remove x.
If the table is full, which means there is no end of the cluster of x, we manually set the element before x as the last element in the cluster and follow the above strange.

4 Hashing with Relocation (24 points)

Hash table is an efficient data structure in general, but it could have poor performance in worst case. After reading hundreds of papers about hash table, Ssy propose a new way to improve the current strategies. In the hash table he proposed, every element is stored along with its probe sequence lengths (PSL). The PSL of an element is defined as the difference between its desired bucket and the bucket it stays. For example, an element x is stored in the hash table, and $h(x) = 2$. But due to collision, it is stored at $hashtable[5]$, then PSL for x is $PSL = 5 - 2 = 3$.
For **insert()**, if the bucket that the key hashes to is empty, we can simply insert the key. If it is occupied, we start linear probing. When encountering an occupied bucket, we compute PSL the new key would have if inserted in that bucket, compare it with PSL of the existing key. If PSL of the new key is greater than the existing key's, we will insert the new key to this bucket and the existing key will be taken out to insert with the same process. Otherwise, we will probe the next bucket. This whole process ends when we encounter an empty bucket.

4.1 Optimized find() (9 points)

The simplest way of $find()$ is to check the bucket to which the key hashes. If the key is not found and the bucket is occupied, we start linear probing. However, this strategy is not efficient enough. Base on the strategy of $insert()$ mentioned above, propose a more efficient way to implement $find()$.

Solution: Do not +1 for each try. Instead, keep the average PSL d and check $h(t) + d, h(t) + d - 1, h(t) + d + 1, h(t) + d - 2, h(t) + d + 2, \cdots$ to find the key.

4.2 Optimized remove() (9 points)

As simply marking bucket as *deleted* during **remove()** will cause poor performance, please propose a better strategy to implement **remove()** in this special hash table.
Hints: make use of PSLs, the main idea is similar to **insert()**.

Solution: To remove x at a, first we remove x. Then, start from $a + 1$,if it not empty and $PSL > 0$, shift to a. Repeat the above until we reach a empty bucket or $PSL == 0$.

4.3 Other Improvements? (6 points)

Explain why this special hash table has a better performance than the one we talked in the lectures. Can you think of other improvements can be done to improve the performance? Think about how to avoid worst case to happen, answer this question briefly in 2-3 sentences.

Solution: It makes the number of tries in $find()$ minimize by calculating and storing PSLs.
We can store the at-least nearest insert place for each bucket when it is inserted and then update it when this bucket is visited later.
The number comes from the return value of function that successfully insert new value since new values must be inserted at the at-least nearest place.
But it do not need to update every bucket before the place to save time.
So the value may not be accurate but can provide at-least end place.
When inserting, instead of check buckets one by one, go to the end directly, which can save time.
When remove, update the values of buckets before it until reach a empty bucket to make sure it provide at-least nearest end place.

5 Longest Probing Sequence (20 points)

When we are using linear probing, it is very easy to form large clusters in our hash table. These clusters can affect the performance of our program greatly in turn. So we want to find the expected longest probing sequence and go deep into analysis of linear probing. Finish the four steps below. Suppose we have a hash table of size m with $n \leq m/2$ elements.
a. Assuming uniform hashing, show that for $i = 1, 2, \ldots, n$, the probability is at most 2^{-k} that the i th insertion requires strictly more than k probes.
b. Show that for $i = 1, 2, \ldots, n$, the probability is $O\left(1/n^2\right)$ that the i th insertion requires more than $2 \lg n$ probes.
Let the random variable X_i denote the number of probes required by the i th insertion. You have shown in part (b) that $\Pr\left\{X_i > 2 \lg n\right\} = O\left(1/n^2\right)$. Let the random variable $X = \max_{1 \leq i \leq n} X_i$ denote the maximum number of probes required by any of the n insertions.
c. Show that $\Pr\{X > 2 \lg n\} = O(1/n)$.
d. Show that the expected length $\mathrm{E}[X]$ of the longest probe sequence is $O(\lg n)$.
Hints: Recall the formula for expectation is $E[X] = \sum_k P[X = k] \cdot k < P[X \leq k] \cdot k + P[X > k] \cdot X_{max}$.

Solution: